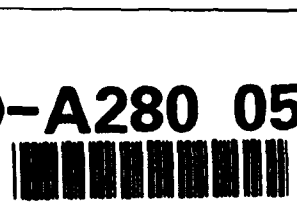


$O(\log^2 n)$ Time Efficient Parallel Factorization of Dense, Sparse Separable, and Banded Matrices

John H. Reif

April 11, 1994
CMU-CS-94-113

Carnegie Mellon

0

$O(\log^2 n)$ Time Efficient Parallel Factorization of Dense, Sparse Separable, and Banded Matrices

John H. Reif

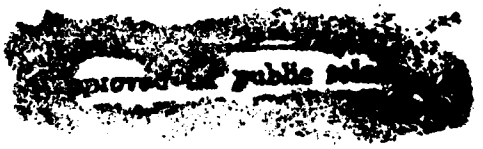
April 11, 1994
CMU-CS-94-113

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213-3890

To appear in *6th Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA'94), New Jersey, July 1994.

Author's permanent address: Department of Computer Science, Duke University, Durham, NC 27708-0129.

Supported by ARPA/ISTO Grant N00014-91-J-1985, Subcontract KI-92-01-0182 of ARPA/ISTO prime Contract N00014-92-C-0182, NSF Grant NSF-IRI-91-00681, NASA subcontract 550-63 of prime Contract NAS5-30428.



Keywords: Parallel algorithms, linear systems, LU factorization, dense matrices, sparse matrices, Newton iteration, banded matrices

Abstract

Known polylog parallel algorithms for the solution of linear systems and related problems require computation of the characteristic polynomial or related forms, which are known to be highly unstable in practice. However, matrix factorizations of various types, bypassing computation of the characteristic polynomial, are used extensively in sequential numerical computations and are essential in many applications.

This paper gives new parallel methods for various exact factorizations of several classes of matrices. We assume the input matrices are $n \times n$ with either integer entries of size $\leq 2^{n^{O(1)}}$ or rational entries expressible as a ratio of integers of size $\leq 2^{n^{O(1)}}$. We make no other assumption on the input. We assume the arithmetic PRAM model of parallel computation. Our main result is a reduction of the known parallel time bounds for these factorizations from $O(\log^3 n)$ to $O(\log^2 n)$. Our results are work efficient: we match the best known work bounds of parallel algorithms with polylog time bounds, and are within a $\log n$ factor of the work bounds for the best known sequential algorithms for the same problems.

The exact factorizations we compute for symmetric positive definite matrices include: recursive factorization sequences and trees, LU factorizations, QR factorizations, and reduction into upper Hessenberg form.

The classes of matrices for which we can efficiently compute these factorizations include:

1. dense matrices, in time $O(\log^2 n)$ with processor bound $P(n)$ (the number of processors needed to multiply two $n \times n$ matrices in $O(\log n)$ time),
2. block diagonal matrices, in time $O(\log^2 b)$ with $P(b)n/b$ processors,
3. sparse matrices which are $s(n)$-separable (recursive factorizations only), in time $O(\log^2 n)$ with $P(s(n))$ processors where $s(n)$ is of the form n^γ for $0 < \gamma < 1$, and
4. banded matrices, in parallel time $O((\log n)\log b)$ with $P(b)n/b$ processors.

Our factorizations also provide us similarly efficient algorithms for exact computation (given arbitrary rational matrices that need not be symmetric positive definite) of the following: solution of the corresponding linear systems, the determinant, the inverse.

Thus our results provide the first known efficient parallel algorithms for exact solution of these matrix problems, that avoids computation of the characteristic polynomial or related forms. Instead we use a construction which modifies the input matrix, which may initially have arbitrary condition, so as to have condition nearly 1, and then applies a multilevel, pipelined Newton iteration, followed by a similar multilevel, pipelined Hensel Lifting.

1 Introduction

1.1 Assumptions and Machine Model

For our model of computation, we assume the algebraic Parallel Random Access Machine (PRAM) where each arithmetic or logical operation such as addition, subtraction, multiplication, division, and comparison can be done in one step by a given processor. Processors can execute such operations in parallel. Our time complexity bounds are based on arithmetic complexity, that is the number of these parallel steps. We assume the $n \times n$ matrices input to our algorithms have integer entries of size $\leq 2^{n^{O(1)}}$ or rational entries expressible as ratio of integers of size $\leq 2^{n^{O(1)}}$. The matrices may be dense, sparse separable, or banded.

Let $P(n) = n^{\omega}$ be the minimum number of PRAM processors necessary to multiply two $n \times n$ matrices in $O(\log n)$ parallel steps. (Currently, $\omega = 2.376$ and we assume $\omega > 2$.)

1.2 Motivation

Many problems in engineering and science rely on the solution of linear systems. As the problem size grows, the resulting linear systems can grow to enormous size, and can, in turn, require very large computational effort to solve. This motivates both the search for work efficient and simultaneous parallel time fast algorithms.

One of the success stories in the field of computer science algorithms and numerical analysis is the development of efficient sequential algorithms for rapid solution of linear systems. Previous researchers have exploited sparsity and/or the structure of the linear system to improve these computations. In practice, the use of parallel processing can give a further increase in speed. However, there remains a considerable discrepancy between the theoretical methods proposed for parallel solution of linear systems and the methods that are actually used. This is the main motivation of our paper.

Pan and Reif [PR 85,PR 89] show that the inverse of a well conditioned nonsingular $n \times n$ dense integer or rational matrix, and the solution of the corresponding linear system, can be approximately computed with high accuracy by Newton iterations in parallel time $O(\log^2 n)$ with $P(n)$ processors, without construction of the characteristic polynomial. Subsequently Pan and others [Pa 85,Pan 87] showed that the inverse of an arbitrary (not necessarily well conditioned) nonsingular $n \times n$ dense integer or rational matrix A can be exactly computed in parallel time $O(\log^2 n)$ with $P(n)$ processors, using a reduction to the computation of the characteristic polynomial or a related form. (Similar reductions to can also be made for matrices over an arbitrary fields [KP 91,KP 92,J 92].) Thus the linear system $Ax = b$ can theoretically be exactly solved within this complexity by the computation $x = A^{-1}b$, where A^{-1} is the matrix inverse.

However, numerical analysis practitioners certainly would generally not solve linear systems by construction of the characteristic polynomial or a related form, which is known to be numerically unstable. Many practicing numerical analysts would object to this approach, for well known stability considerations.

Instead, they generally much prefer to factorize the matrices, and solve triangular linear systems. For example, suppose we are given a nonsingular $n \times n$ matrix A. If A is symmetric positive definite, then A will be factored $A = LU$ where $U = L^T$(also known in this case as a Cholesky factorization), where L is nonsingular lower triangular and U is nonsingular upper triangular.

The requirement that the matrix be ***symmetric positive definite*** presents no difficulties, as we now show. Even if A is not symmetric positive definite, then A^TA is, so A^TA can be factored as $A^TA = LU$ (this squares the condition number, which may be problem if A is badly conditioned). With this preprocessing, and given an n-vector b, the linear system $Ax = b$ can be solved in two back-solving stages by first solving for n-vector y and then for n-vector z in the triangular linear systems $Ly = b, Uz = y$ (For which there are well known highly efficient parallel algorithms; see JáJá [J 92]). Then if $A = LU$ then we can let $x = z$, and otherwise if $A^TA = LU$ then we can let $x = Az$.

Known efficient parallel algorithms for exact computation of the determinant of A (see [Pa 85,Pan 87,KP 91,KP 92,J 92]) would also require the computation of the characteristic polynomial of A. In contrast, given the LU decomposition as above, if $A = LU$, then $det(A) = det(L)det(U)$, and otherwise if $A^TA = LU$, then $det(A)^2 = det(A^TA) = det(L)det(U)$. In either case, the determinants of these triangular matrices are obtained by multiplying all the elements of their principal diagonals.

Furthermore, singular value decompositions and eigenvalue computations generally begin with QR factorization which is computable from the LU factorization. Eigenvalue computations generally use a further reduction to upper Hessenberg form computed from the QR factorization (see Golub and van Loan [GL 83]). Thus matrix factorizations of various types are used extensively in numerical computations and are essential in many applications.

For dense matrices, known efficient parallel algorithms for exact LU and QR factorization and reduction to upper Hessenberg form ([Pan 87]) cost $O(\log^3 n)$ time and also require the computation of the characteristic polynomial of A. The objective of this paper is to provide efficient parallel algorithms to factor matrices in time $O(\log^2 n)$ with $P(n)$ processors, without computation of the characteristic polynomial.

1.3 Our Parallel Algorithms for Dense, Sparse and Banded Matrices

The general technique of approximate solution, followed by Hensel Lifting, has a long history in numerical and algebraic computation. For a recent example see Pan [Pa 85,Pan 87]. We use such a construction which modifies the input

matrix (which initially may have arbitrary condition, so could be very badly conditioned), so that the resulting matrix is extremely diagonally dominant and has condition nearly 1. Explicit computation of matrix inverse of badly conditioned matrices is thus avoided, and instead we approximate the inverse of diagonally dominant matrices. To compute the recursive factorization, we apply a multilevel, pipelined Newton iteration, followed by Hensel Lifting. Our algorithms give an exact factorization, but nevetheless avoid computation of the characteristic polynomial or related forms.

Using reductions to the recursively factorization algorithm, we exactly compute, for symmetric positive definite matrices, LU and QR factorizations, and also compute their reduction into upper Hessenberg form. Using further reductions to the LU factorization, we exactly compute, for arbitrary integer or rational matrices (which need not be symmetric positive definite) solution of the corresponding linear systems, the determinant, the inverse.

1. For dense matrices, we show that all of these factorizations and computations can be done in parallel time $O(\log^2 n)$ with $P(n)$ processors.

2. For sparse matrices, (with $O(n)$ non-zero entries) which are $s(n)$-*separable* (i.e. the separator size of the sparsity graph of the matrix is of size $s(n)$), we show LU and QR factorizations can be done in parallel time $O(\log^2 n$ with $P(s(n))$ processors where $s(n)$ is of the form n^γ for $0 < \gamma < 1$.

3. For b-block diagonal matrices (consisting of a sequence of $b \times b$ blocks on the diagonal) we show LU and QR factorizations can be done in parallel time $O(\log^2 b)$ with $P(b)n/b$ processors.

4. For b-banded matrices, (where the non-zeros occur within only a band of width b around the diagonal) we show LU and QR factorizations can be done in parallel time $O(\log n \log b)$ with $P(b)n/b$ processors.

Previous work of Pan [Pan 87] gave parallel time $O(\log^3 n)$ with $P(n)$ processors for factoring dense symmetric positive definite matrices, including LU and QR factorizations, and also computing their reduction into upper Hessenberg form. Pan shows the solution and determinant of b-banded matrices can be computed in parallel time $O(\log n \log b)$ with $P(b)n/b$ processors, but his results do not extend to LU or QR factorizations. Armon and Reif [AR 92] gave parallel time $O(\log^2 n)$ with $P(s(n))^{1+\epsilon}$ processors, for $\epsilon > 0$, for recursive factorization (but not LU or QR factorizations) of nonsingular symmetric positive definite matrices.

1.4 Organization of the Paper

In Section 1, we have motivated the problems we solve and stated our results and previous results. Section 2 is a preliminary. In this section, we define matrix notations, and problems, as well as introduce the Recursive Factorization

(RF) Sequence and Tree of matrices, and show that matrix problems can be efficiently reduced to RF computation in $O(log^2 n)$ parallel time using $P(n)$ processors. Section 3 gives the parallel algorithm to compute the RF Tree of a matrix with parallel time $O(log^2 n)$ using $P(n)$ processors. Details of the two components of the algorithm, namely Newton-Hensel Lifting and Newton Iteration are left out of Section 3 and are dealt with in detail in Section 4 and Section 5 separately. Section 6 extends these results to parallel nested dissection, giving efficient parallel factorizations of symmetric matrices with $s(n)$-separable graphs in $O(log^2 n)$ time and $P(s(n))$ processors. Section 7 discusses banded matrices.

2 Preliminaries

This section contains some definitions, notations and previously known results that will be of use later in this paper.

2.1 Matrix Definitions and Notation

2.1.1 Matrix Assumptions

All matrix products are inner products. Vectors and matrices will be denoted by lower and upper case characters, respectively.

Generally, we assume input A is a square matrix of size $n \times n$, except where we compute QR-factors, where the input A can be a rectangular matrix of size $m \times n$, where $m \geq n$. We assume throughout this paper, without loss of generality, that n is a power of 2. Although all our results apply to rational matrices, note that we can always multiply a rational matrix by an appropriate integer to form an integer matrix. For simplicity, we assume without loss of generality throughout this paper the matrices *input* to our factorization algorithms have integer entries of size $< 2^{n^{O(1)}}$. However, our algorithms will in general generate and output rational matrices.

Throughout this paper all logarithms are base 2.

2.1.2 Matrix Definitions and Specialized Matrices

I and O will denote identity and null matrices of the appropriate sizes. A^T will denote the transpose of a matrix A. A is define to be *symmetric* if $A^T = A$. Let $det(A)$ denote the *determinant* of A. A is *singular* if $det(A) = 0$, else it is *nonsingular*. If A is nonsingular, then the *inverse* A^{-1} is defined and the *adjoint* is the matrix adj(A) =det$(A)A^{-1}$ (the adjoint is an integer matrix if A is).

Let $A = [a_{ij}]$ denote the elements of A. The *principal diagonal* of A are the elements $a_{11}, a_{22}, \ldots, a_{nn}$. A *minor* of A is a sub-matrix induced by a sequence of rows, say $i_1, i_2, \ldots, i_{n'}$ and columns, say $j_1, j_2, \ldots, j_{m'}$. A *principal minor* of A is a square minor induced by selecting the same sequence of row

indices, say $i_1 = j_1, i_2 = j_2, \ldots, i_{n'} = j_{n'}$ as column indices, and it is a *leading principal minor* of A if these indices are consecutive starting at 1, say $i_1 = j_1 = 1, i_2 = j_2 = 2, \ldots, i_{n'} = j_{n'} = n'$. A is *positive definite* if $x^T Ax > 0$ for all nonzero vectors x. If A is positive definite, every principal minor is also positive definite. If A is positive definite, then A is always nonsingular. For any nonsingular matrix A, $A^T A$ is symmetric positive definite.

A is *orthogonal* if $A^T A = I$. A is *lower (upper) triangular* if $a_{ij} = 0$ for $i < j$ ($j < i$, respectively). A is *b-banded* if $a_{ij} = 0$ for $2|j - i| + 1 > b$, so the non-zeros occur within only a band of width b around the diagonal. A is *b-block diagonal* if A has nonzero entries only at a sequence of disjoint $b \times b$ blocks on the diagonal.

We leave to Subsection 2.4 a definition of various matrix factorizations.

2.1.3 Matrix Norms

Let the elements of matrix $A = [a_{ij}]$. For $p = 1, 2, \infty$, let $\|A\|_p = \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$ denote the p-norm of matrix A, so $\|A\|_1 = \max_j \sum_i |a_{ij}|$ and $\|A\|_\infty = \max_i \sum_j |a_{ij}|$.

For each such $p = 1, \infty$, (see Golub and van Loan [GL 83]) $\|A\|_p / n \leq \max_{i,j} |a_{i,j}| \leq \|A\|_2$, for $A = [a_{ij}]$. Also, for each such $p = 1, \infty$, (see Golub and van Loan [GL 83], p. 57) $\|A\|_p / \sqrt{n} \leq \|A\|_2 \leq \|A\|_p \sqrt{n}$. We can bound $det(A) \leq (n\|A\|_2)^n$.

If A is nonsingular, let $cond_p(A) = \|A\|_p \cdot \|A^{-1}\|_p$ for $p = 1, 2, \infty$, and let $cond(A) = \|A\|_2 \|A^{-1}\|_2$. Note that for nonsingular A, $cond(A) = \|A\|_2 \cdot \|A^{-1}\|_2 \geq 1$, so $\|A^{-1}\|_2 \geq 1/\|A\|_2$. A is *well conditioned* if $cond(A) \leq n^{O(1)}$. Note that if A is well conditioned, then so is $A^T A$.

Note: throughout this paper, *we drop the subscript p in the case we wish to indicate the 2-norm*, so $\|A\| = \|A\|_2$.

2.2 Recursive Factorization (RF) Sequences of Matrices

Recursive Factorization (RF) SEQUENCE Problem: If possible, compute a sequence of matrices $A = A_0, A_1, \ldots, A_{\log n}$ where for $d = 0, 1, \ldots, \log n - 1$, A_d is an $n/2^d \times n/2^d$ matrix which is partitioned as

$$A_d = \begin{bmatrix} W_d & X_d \\ Y_d & Z_d \end{bmatrix},$$

where W_d, X_d, Y_d, Z_d are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and $A_{d+1} = Z_d - Y_d W_d^{-1} X_d$ is called the *Schur complement*.

If, for any d, no such factorization is possible, then A has no RF Sequence. However, any symmetric positive definite matrix has a unique RF sequence (see Golub and van Loan [GL 83] and Pan [Pan 87]).

Proposition 2.1 *(follows from known properties of Schur complements and Pan and Reif [PR 85, PR 92]) In the RF sequence, for all* $d, 0 \leq d \leq \log n - 1$ *and* $\alpha \in \{0, 1\}^d$,

- *if A_d is symmetric positive definite, then so are A_{d+1} and W_d,*
- $\|A_{d+1}\|, \|W_d\| \leq \|A_d\|$,
- $\|A_{d+1}^{-1}\|, \|W_d^{-1}\| \leq \|A_d^{-1}\|$,
- $1/\|A_d^{-1}\| \leq \min(\|A_{d+1}\|, \|W_d\|)$,
- $1/\|A_d\| \leq \min(\|A_{d+1}^{-1}\|, \|W_d\|^{-1})$,
- $cond(W_d) \leq cond(A)$.

This RF sequence gives the following useful recursive formulae:

Proposition 2.2 *The LU factorization of A can be recursively computed from the RF sequence of A, as follows:*

$$A_d = \begin{bmatrix} I & O \\ Y_d W_d^{-1} & I \end{bmatrix} \begin{bmatrix} W_d & O \\ O & A_{d+1} \end{bmatrix} \begin{bmatrix} I & W_d^{-1} X_d \\ O & I \end{bmatrix}$$

Proposition 2.3 *The inverse of A can be recursively computed from the RF sequence of A, as follows:*

$$A_d^{-1} = \begin{bmatrix} I & -W_d^{-1} X_d \\ O & I \end{bmatrix} \begin{bmatrix} W_d^{-1} & O \\ O & A_{d+1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_d W_d^{-1} & I \end{bmatrix}.$$

The RF sequence provides a divide and conquer technique used in many theoretically efficient sequential algorithms for matrix inverse. (see [AHU 74]). For example, the Strassen block matrix algorithm shows that computing the inverse of two $n \times n$ matrices that are partitioned into four blocks (each of size $n/2 \times n/2$) reduces to matrix inverses and products on the block submatrices. (see [AHU 74]). The above formula expresses the inverse of the input matrix in terms of a constant number products, sums and two inverses of four $n/2 \times n/2$ submatrices.

2.3 Recursive Factorization (RF) Trees of Matrices

RF TREE Problem: If possible, compute a full binary tree of depth $\log n$ whose nodes are matrices. All nodes of depth $d, 0 \leq d \leq \log n$ are $n/2^d \times n/2^d$ matrices with notation $A_{d,\alpha}$ where $\alpha \in \{0,1\}^d$ is a binary string of length d.

If $d = 0$ then $A_{d,\alpha} = A$ is the root of the tree and α is the empty string. For $0 \leq d < \log n$, each matrix $A_{d,\alpha}$ of depth d has exactly two children in the tree, $A_{d+1,\alpha 1}$ and $A_{d+1,\alpha 0}$ of depth $d+1$ which will be defined by recursion. In particular, for $d = 0, 1, \ldots, \log n - 1$, $A_{d,\alpha}$ is an $n/2^d \times n/2^d$ matrix which is partitioned as

$$A_{d,\alpha} = \begin{bmatrix} A_{d+1,\alpha 0} & X_{d,\alpha} \\ Y_{d,\alpha} & Z_{d,\alpha} \end{bmatrix}$$

where $A_{d+1,\alpha 0}, X_{d,\alpha}, Y_{d,\alpha}, Z_{d,\alpha}$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and $A_{d+1,\alpha 1} = Z_{d,\alpha} - Y_{d,\alpha} A_{d+1,\alpha 0}^{-1} X_{d,\alpha}$ is the *Schur complement*. If, for any d, no such factorization is possible, then A has no RF tree.

Important Note: The RF tree of A is very similar to the previously defined RF sequence $A = A_0, A_1, A_2, \ldots, A_{\log n}$. In particular, for each $d = 1, \ldots, \log n$ $A_d = A_{d,\alpha}$, where $\alpha = 1^d$. The only difference is that the RF tree also recursively factors each of the $W_d = A_{d+1,\alpha 0}$ matrices appearing in the RF sequence.

Any symmetric positive definite matrix has an RF sequence and therefore an RF tree, and it is unique.

This RF tree gives the following useful recursive formulae:

Proposition 2.4 *The LU factorization of A can be recursively computed from the RF tree of A, as follows:* $A_{d,\alpha} =$

$$\begin{bmatrix} I & O \\ Y_{d,\alpha} A_{d+1,\alpha 0}^{-1} & I \end{bmatrix} \begin{bmatrix} A_{d+1,\alpha 0} & O \\ O & A_{d+1,\alpha 1} \end{bmatrix} \begin{bmatrix} I & A_{d+1,\alpha 0}^{-1} X_{d,\alpha} \\ O & I \end{bmatrix}$$

Proposition 2.5 *The inverse of A can be recursively computed from the RF tree of A, as follows:* $A_{d,\alpha}^{-1} =$

$$\begin{bmatrix} I & -A_{d+1,\alpha 0}^{-1} X_{d,\alpha} \\ O & I \end{bmatrix} \begin{bmatrix} A_{d+1,\alpha 0}^{-1} & O \\ O & A_{d+1,\alpha 1}^{-1} \end{bmatrix} \begin{bmatrix} I & O \\ -Y_{d,\alpha} A_{d+1,\alpha 0}^{-1} & I \end{bmatrix}$$

Note: throughout the rest of this paper, we will deal only with RF trees. Thus we will hereafter simply call an RF tree an *RF*, and we will call the RF tree problem simply the RF problem unless otherwise indicated.

2.4 Reduction of Matrix Problems to RF Computation

In this section we define various matrix problems and their efficient parallel reduction to RF Computation (also, see Pan, p.69 [Pan 87]). We will consider a reduction to be an *efficient parallel reduction* if it can be done in $O(\log^2 n)$ parallel time using $P(n)$ processors.

1. *LU* FACTORIZATION (if A is symmetric, then $U = L^T$ and this problem is known as CHOLESKY FACTORIZATION): If possible, factor $A = LU$ where L is nonsingular lower triangular, and U is nonsingular upper triangular, otherwise output NO LU FACTORIZATION.

   If A is symmetric positive definite, then A always has a *LU* factorization.

   Note: If A has an RF, then the *LU* factorization can be computed by $O(\log n)$ stages of matrix multiplication using the above recursive formula.

   Lemma 2.1 *There is an efficient parallel reduction from LU FACTORIZATION to the RF problem.*

2. DETERMINANT: Compute $det(A)$.

   Note: If A has a factorization $A = LU$, then $det(A) = det(L)det(U)$. The determinant of a triangular matrix is obtained by multiplying all the elements on its principal diagonal. (This can be computed in $O(\log n)$ time and $n/\log n$ processors by using a balanced binary multiplication tree of size $O(n/\log n)$ whose leaves have $\log n$ elements each.) So $det(L) = \prod_{i=1}^{n} L_{ii}$ and $det(U) = \prod_{i=1}^{n} U_{ii}$. Otherwise, $A^T A$ is symmetric positive definite, and so has a factorization $A^T A = LU$. Then since $det(A) = det(A^T)$, we have $det(A)^2 = det(A^T A) = det(L)det(U)$.

   Lemma 2.2 *There is an efficient parallel reduction from DETERMINANT to the RF problem.*

3. INVERSE: If A is nonsingular, then compute $A^{-1} = \frac{adj(A)}{det(A)}$, where $adj(A)$ is the adjoint matrix of A, otherwise output SINGULAR.

   Note: If A has an RF, then A^{-1} can be computed by the above RF sequence or tree formula. Otherwise, $A^T A$ is symmetric positive definite, so if A is nonsingular then A has an RF. Then $(A^T A)^{-1} = (A)^{-1}(A^T)^{-1}$, can be computed by the above RF tree formula, and we can compute $A^{-1} = A((A)^{-1}(A^T)^{-1}) = A(A^T A)^{-1}$ by one more matrix product.

   Lemma 2.3 *INVERSE has an efficient parallel reduction to the RF problem.*

4. LINEAR SYSTEM SOLVE: If A is nonsingular, then compute $A^{-1}v$, otherwise output SINGULAR.

   Here we can apply the efficient parallel reduction given in our Subsection 1.2 to LU factorization of $A^T A$ (which we can compute by Lemma 2.1 by efficient parallel reduction to the RF.), and to solution of triangular linear systems, for which there are known efficient parallel algorithms (see JáJá [J 92]).

   Lemma 2.4 *There is an efficient parallel reduction from LINEAR SYSTEM SOLVE to the RF problem.*

5. QR FACTORIZATION: If possible, factor $m \times n$ matrix $A = QR$ where R is a nonsingular upper triangular matrix and Q is an orthogonal matrix (so $Q^T Q = I$) of size $m \times n$, for $m \geq n$. If the QR factorization is not possible, then output NO QR FACTORIZATION (Rank deficient).

   Note: the QR-factors of A can be computed from the LU-factors of $A^T A$, where $R = U$ and $Q = A \cdot R^{-1}$.

Lemma 2.5 *There is an efficient parallel reduction from QR FACTORIZATION to the RF problem.*

6. HESSENBERG REDUCTION: Compute a matrix $H = Q^T AQ$ having upper Hessenberg form $H_{ij} = 0$ if $i > j + 1$, and compute Q, which is an orthogonal matrix. Note: if A is symmetric, then H is tridiagonal.

   Note: The *Krylov matrix* of an $n \times n$ matrix A and n-vector v, is an $n \times m$ matrix $K(A, v) = (v, Av, A^2v, \ldots, A^{m-1}v)$. Borodin and Munro ([BM 75]p.128) describe a well known algorithm for the Krylov matrix requiring $2 \log m$ multiplications of matrices of size at most $n \times \max(n, m)$. They observe that the matrix powers $A, A^2, \ldots, A^{2^{\lfloor \log m \rfloor}}$ can be computed in $\lfloor \log m \rfloor$ stages of matrix products, and that $K(A, v)$ can be computed in $\log m$ further stages, where using the identity for $i = 1, \ldots, \lfloor \log m \rfloor$, $(A^{2^i}v, Av, A^2v, \ldots, A^{2^{i+1}}v) = A^{2^i}(v, Av, A^2v, \ldots, A^{2^i}v)$.

   Further Note: If $K(A, v)$ is nonsingular with QR factorization, $K(A, v) = QR$, then $H = Q^T AQ$ is in upper Hessenberg form.

Lemma 2.6 *There is an efficient parallel reduction from HESSENBERG REDUCTION to the RF problem.*

3 Our Parallel Algorithm for Computing an RF of a Matrix

Fix a nonsingular $n \times n$ matrix A, with integer entries of size $\leq 2^{n^{O(1)}}$.

3.1 Deterministic Choice of Modulus

Proposition 3.1 *Let p be a prime of size at least $(n\|A\|)^{nc_0}$ for some $c_0 > 2$. If $det(A) \neq 0$, then $0 \not\equiv det(A) \pmod{p}$*

Also, if $det(K(A, v)) \neq 0$, then $0 \not\equiv det(K(A, v)) \pmod{p}$.

3.2 Random Choice of Modulus

Proposition 3.2 *(See Schwartz [Sc 80] and Pan [Pan 87]) Let p be a random prime of the interval $[2(n\|A\|)^{c_0}/n, 2(n\|A\|)^{c_0}]$, for any $c_0 > 2$. If $det(A) \neq 0$, then $0 \not\equiv det(A) \pmod{p}$ with probability $\geq 1 - \Omega(n \log(n\|A\|)/(n\|A\|)^{c_0})$.*

Also, if $det(K(A, v)) \neq 0$, then $0 \not\equiv det(K(A, v)) \pmod{p}$ with probability $\geq 1 - \Omega(n^2 \log(n\|A\|)/(n\|A\|)^{c_0})$.

3.3 RF Algorithm

We now describe our algorithm:

Algorithm RF

INPUT: an $n \times n$ integer matrix A, with integer entries of size $\leq 2^{n^{O(1)}}$.

Comment: Since we can bound $det(A) \leq 2^{n^{O(1)}}$, all rational quantities in the RF can be expressed as a quotient of two integers of size $\leq 2^{n^{O(1)}}$.

1. Depending on whether a deterministic or randomized algorithm is desired, do steps (a) or (b).

   (a) Let p be any prime from the interval $(n\|A\|)^{nc_0} < p < (n\|A\|)^{nc_0'}$ for some $c_0, c_0' > 2$.

   Comment: By Proposition 3.1, if $det(A) \neq 0$ then $0 \not\equiv det(A) \pmod{p}$.

   (b) Let p be a random prime from the interval $2(n\|A\|)^{c_0}/n < p < 2(n\|A\|)^{c_0}$, for some $c_0 > 2$.

   Comment: By Proposition 3.2, if $det(A) \neq 0$ then $0 \not\equiv det(A) \pmod{p}$ with high likelihood.

2. Let $\bar{A} = A + Ip(n\|A\|_\infty)^{nc_1}$, for a sufficiently large constant $c_1 > 8$.

   Comment: $\bar{A}$ is extremely diagonally dominant, and thus nonsingular, and $\text{cond}(\bar{A}) \leq 1 + \frac{1}{(n\|A\|)^{\Omega(n)}}$.

   Comment: We can show we actually need to add a much smaller quantity to A, thus decreasing the bit complexity of our algorithm.

   Comment: Since $\bar{A} - A = Ip(n\|A\|_\infty)^{nc_1}$ is divisible by p, $\bar{A} \equiv A \bmod p$, and thus $(\bar{A})^{-1} \equiv A^{-1} \bmod p$.

3. Apply Newton Iteration of Section 5 (Lemma 5.4) to compute in time $O(\log^2 n)$ using $P(n)$ processors, an approximate RF of $\bar{A}$ to accuracy 2^{-n^c}, for a sufficiently large constant $c > 2$.

   Comment: This approximate RF gives for each $\bar{A}_{d,\alpha}$, for $d = 0, \ldots, \log n$ and $\alpha \in \{0,1\}^d$, an approximation of $(\bar{A}_{d,\alpha})^{-1}$ and the LU factorization of $\bar{A}_{d,\alpha}$ to accuracy 2^{-n^c}.

4. For each $d = 0, \ldots, \log n$ and $\alpha \in \{0,1\}^d$ do in parallel

   (a) By applying Lemma 2.2 (which showed DETERMINANT has an efficient parallel reduction to RF), compute in time $O(\log^2 n)$ using $P(n)$ processors an approximation to $det(\bar{A}_{d,\alpha})$ to accuracy within 2^{-n^c} from the approximate LU factorization of $\bar{A}_{d,\alpha}$.

   Comment: Since $det(\bar{A}_{d,\alpha}) \leq (n\|A\|)^n$, and each entry in the approximation can be in error by at most $n^2 2^{-n^c}$, the total error is $(n\|A\|)^n n^2 2^{-n^c} < 1/2$, with choice of sufficiently large constant $c > 2$.

(b) Round this approximation to the nearest integer to compute $det(\bar{A}_{d,\alpha})$ exactly.

(c) If $det(\bar{A}_{d,\alpha}) \equiv 0 \pmod{p}$ for any d, α then exit and **OUTPUT** NO RF of A.

Comment: In this case $det(A_{d,\alpha}) = 0$ implying $A_{d,\alpha}$ is singular, so A has no RF.

(d) Multiply this exact value of $det(\bar{A}_{d,\alpha})$ by the approximation of $(\bar{A}_{d,\alpha})^{-1}$ to get an approximation of the integer adjoint matrix $adj(\bar{A}_{d,\alpha})$ to accuracy within 2^{-n^c}.

(e) Round this approximation to the nearest integer to compute $adj(\bar{A}_{d,\alpha})$ exactly, and thus the exact rational value of $(\bar{A}_{d,\alpha})^{-1} = \frac{adj(\bar{A}_{d,\alpha})}{det(\bar{A}_{d,\alpha})}$.

Comment: We now have the exact RF of $\bar{A}$.

Comment: Recall $\bar{A} \equiv A \bmod p$, so the RF $\pmod{p}$ of A is identical to the RF $\pmod{p}$ of $\bar{A}$.

5. Reduce $\pmod{p}$ the exact RF of $\bar{A}$, yielding the RF $\pmod{p}$ of A

6. Apply Newton-Hensel Lifting of Section 4 (Lemma 4.1) to compute in time $O(\log^2 n)$ using $P(n)$ processors the RF $\pmod{p^{n^c}}$ of A, which is the same as the exact RF of A.

OUTPUT The exact RF of A.

Note that by Lemmas 5.4, 2.2, and 4.1, each major stage of the above RF algorithm takes at most time $O(\log^2 n)$ using $P(n)$ processors. Since there are only 7 such major stages, we have:

Theorem 3.1 *Our algorithm for the exact RF takes parallel time* $O(\log^2 n)$ *using* $P(n)$ *processors.*

By Theorem 3.1 and the efficient parallel reductions of Subsection 2.4, we have the following new results:

Corollary 3.1 *The following problems can be exactly solved in parallel time* $O(\log^2 n)$ *using* $P(n)$ *processors:*

- *LU FACTORIZATION*
- *QR FACTORIZATION.*

Note: the RF will still be constructed for input which are non-symmetric positive definite matrices, so symmetry of the input matrix is not essential (it just simplifies the analysis (Lemma 5.4) of the pipelined Newton iterations described in Section 5). However, the RF algorithm on an input matrix which is singular will not result in a complete RF.

By a slight modification of the above construction (see Pan [Pan 87]), we can compute a HESSENBERG REDUCTION. Let $H = [h_{ij}]$ be the $n \times n$ matrix where

$$h_{ij} = \begin{cases} 1 & \text{if } j - i = 1 \bmod n, \\ 0 & \textit{otherwise.} \end{cases}$$

Let $v = [1, 0, \ldots, 0]$ and $A'' = HA + Hp(n\|A\|_\infty)^{nc_1}$, for an appropriate constant $c_1 > 3$ and for a prime p as in the RF algorithm.

Pan [Pan 87] shows that the Krylov matrix $K(A'', v) = [v, A''v, \ldots, A''^{n-1}v]$ is strongly diagonally dominant, and that same proof implies $K(A'', v)$ is extremely diagonally dominant. As Pan [Pan 87] notes, some simple modifications of our RF algorithm allow one to compute the HESSENBERG REDUCTION. We can compute $(H^T A'')^{-1}$ using the RF algorithm, because $H^T A'' = A + Hp(n\|A\|_\infty)^{nc_1}$ is strongly diagonally dominant, and because we have already computed $det(H^T A'') = det(H^T)det(A'') = (-1)^{n-1}det(A'')$. Compute $(H^T A'')^{-1}det(H^T A'') = adj(H^T A'')$. Then we can compute the integer adjoint matrix $adj(A)$ via reduction modulo p and finally compute $A^{-1} = \frac{adj(A)}{det(A)}$.

Corollary 3.2 *HESSENBERG REDUCTION can be exactly solved in parallel time $O(\log^2 n)$ using $P(n)$ processors.*

Note that our parallel time bounds for LU FACTORIZATION, QR FACTORIZATION and HESSENBERG REDUCTION are new. Previously Pan [Pan 87] had proved the same processor bounds for these listed problems with a time bound of $O(\log^3 n)$.

The following additional problems were previously known to be exactly solvable (see [Pa 85,Pan 87,KP 91,KP 92,J 92]) in time $O(\log^2 n)$ using $P(n)$ processors, requiring reduction to the computation of the characteristic polynomial.

- DETERMINANT
- INVERSE
- LINEAR SYSTEM SOLVE.

Our techniques achieve the same bounds, without the need to compute the characteristic polynomial.

3.4 Factorization of Block Diagonal Matrices

Recall that A is *b-block diagonal* if A has nonzero entries only at a sequence of $b \times b$ disjoint blocks on the diagonal.

Corollary 3.3 *If A is b-block diagonal, then the following problems can be exactly solved in parallel time $O(\log^2 b)$ with $P(b)n/b$ processors:*

- *RF*
- *LU FACTORIZATION*
- *QR FACTORIZATION*

4 Newton-Hensel Lifting

4.1 Newton-Hensel Lifting of a Matrix

Fix a nonsingular $n \times n$ matrix A and a prime p. In this section we assume we have already computed $A^{-1} \bmod p$. Zassenhause extended Hensel's lifting to exponentially increase the modulus. The resulting Newton-Hensel Lifting is the following algorithm:

INPUT: a positive number k, and $n \times n$ matrices A and $A^{-1} \bmod p$.

1. $S^{(0)} = A^{-1} \bmod p$
2. For $i = 1, \ldots, k$ do $S^{(k)} = S^{(k-1)}(2I - AS^{(k-1)}) \bmod p^{2^k}$.

Moenck and Carter [MC 79] show

Proposition 4.1 $S^{(k)} = A^{-1} \bmod p^{2^k}$.

4.2 Newton-Hensel Lifting of Modular Recursive Factorizations

Recall that the RF of A is a full binary tree of depth $\log n$. Each internal node is an $n/2^d \times n/2^d$ matrix $A_{d,\alpha}$. The RF of $A_{d,\alpha}$ is defined in terms of the RF of two $n/2^{d+1} \times n/2^{d+1}$ matrices, namely $A_{d+1,\alpha 1}$ and $A_{d+1,\alpha 0}$, and furthermore $A_{d+1,\alpha 1}$ is defined in terms of submatrices of $A_{d,\alpha}$ and the inverse of $A_{d+1,\alpha 0}$.

Let an *RF* $\pmod p$ of an $n \times n$ matrix A be an RF of matrix A where each element is taken $\pmod p$. This will also be called a *modular RF*. In this section we assume we are given an RF $\pmod p$ of matrix A . We shall compute the RF $\pmod{p^{2^k}}$ of A.

Recall that $P(n) = n^\omega$ is the minimum number of PRAM processors necessary to multiply two $n \times n$ matrices in $O(\log n)$ parallel steps, where we assume $\omega > 2$.

Proposition 4.2 $\sum_{d=0}^{\log n} 2^d O(P(n/2^d)) \leq O(P(n))$.

Note that the obvious way to compute the RF $\pmod{p^{2^k}}$ of A is by proceeding level by level through the RF in $\log n$ stages, each requiring $O(k \log n)$ time and $O(P(n)) \geq 2^d O(P(n/2^d))$ processors for the required k matrix products for the nodes of depth d. This requires total parallel time $O(k \log^2 n)$ using $O(P(n))$ processors.

Newton-Hensel Lifting Algorithm for an RF

INPUT: RF $\pmod p$ of A and number $k \geq 1$.

For each d, α for $0 \leq d \leq \log n$ and $\alpha \in \{0,1\}^d$, do in parallel:

1. INITIALIZATION:

   (a) Let $A_{d,\alpha}^{(0)} \equiv A_{d,\alpha} \pmod p$

(b) Let $S^{(0)}_{d+1,\alpha 0} \equiv A^{-1}_{d+1,\alpha 0} \pmod{p}$.

2. For each $i = 1, \ldots, k$ do

Loop Invariant: We have just computed
$A^{(i-1)}_{d,\alpha} \equiv A_{d,\alpha} \bmod p^{2^{i-1}}$ and
$S^{(i-1)}_{d,\alpha} \equiv A^{-1}_{d,\alpha} \bmod p^{2^{i-1}}$.

(a) Let $S^{(i)}_{d,\alpha} = S^{(i-1)}_{d,\alpha}\,(2I - A_{d,\alpha}S^{(i-1)}_{d,\alpha})$.

(b) For $d < \log n$, let

$$A^{(i)}_{d,\alpha} = \begin{bmatrix} A^{(i)}_{d+1,\alpha 0} & X^{(i)}_{d,\alpha} \\ Y^{(i)}_{d,\alpha} & Z^{(i)}_{d,\alpha} \end{bmatrix},$$

where $A^{(i)}_{d+1,\alpha 0}, X^{(i)}_{d,\alpha}, Y^{(i)}_{d,\alpha}, Z^{(i)}_{d,\alpha}$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and
$A^{(i)}_{d+1,\alpha 1} = Z^{(i)}_{d,\alpha} - Y^{(i)}_{d,\alpha}S^{(i)}_{d+1,\alpha 0}X^{(i)}_{d,\alpha}$.

OUTPUT: RF $\pmod{p^{2^k}}$ of A given by the $\{A^{(k)}_{d,\alpha}\}$.

This algorithm, as stated, requires parallel time $O(k \log^2 n)$ using $O(P(n))$ processors. However, we can use the technique of *stream contraction* (see Pan and Reif [PR 91]) to decrease the time to $O((k + \log n)\log n)$ time without a processor penalty. The idea is to note that for each $d, 0 \le d < \log n$, and each $\alpha \in \{0,1\}^d$ defining an internal node $A_{d,\alpha}$ of depth d, the following hold:

1. The matrices $A_{d+1,\alpha 0} \bmod p^{2^{i-1}}$ and $A_{d+1,\alpha 1} \bmod p^{2^{i-1}}$ are defined in terms of submatrices of $A_{d,\alpha} \bmod p^{2^i}$.

2. The computation $S^{(i)}_{d,\alpha} \equiv A^{-1}_{d,\alpha} \bmod p^{2^i}$ depends on the previous computations

(a) $S^{(i-1)}_{d+1,\alpha 1} \equiv A^{-1}_{d+1,\alpha 1} \bmod p^{2^{i-1}}$ and

(b) $S^{(i-1)}_{d+1,\alpha 0} \equiv A^{-1}_{d+1,\alpha 0} \bmod p^{2^{i-1}}$.

This implies that we can pipeline the computation as follows: As our basis step $t = 0$ we have done the computation $S^{(0)}_{d,\alpha}$ for all $d = 0, \ldots \log n$ and each $\alpha \in \{0,1\}^d$. We assume for our induction hypothesis that at time $t, 0 \le t < k + \log n$ we have computed each $S^{(i)}_{d,\alpha}$ for all i, d, α where $0 \le d \le \log n$, $0 \le i \le t - d$ and $\alpha \in \{0,1\}^d$.

Then we apply the RF formula and compute one further product at each node of the recursion tree to compute by time $t + 1$ each $S^{(i+1)}_{d,\alpha}$ for all i, d, α where $0 \le d \le \log n, i = t + 1 - d$ and $\alpha \in \{0,1\}^d$.

Summarizing, we have (using a small constant factor slowdown to reduce the processor bounds from $O(P(n))$ to $P(n)$)

Lemma 4.1 *Given an RF* $(\bmod p)$ *of an* $n \times n$ *matrix* A, *we can compute an RF* $(\bmod p^{2^k})$ *of* A *in parallel time* $O((k+\log n)\log n)$ *using* $P(n)$ *processors.*

5 Newton Iterations for Approximation of an RF of Diagonally Dominant Matrices

5.1 Extremely Diagonally Dominant Matrices

Let A be an $n \times n$ matrix where $A = [a_{ij}]$. A is *ϵ-diagonally dominant* for some $\epsilon > 0$ if for all $i, 1 \leq i \leq n$,

$$\epsilon |a_{ii}| > \sum_{j=1, j \neq i}^{n} |a_{ij}|.$$

Let $D(A) = diag(a_{11}, a_{22}, \ldots, a_{nn})$ denote the $n \times n$ diagonal matrix with the diagonal entries $a_{11}, a_{22}, \ldots, a_{nn}$. Then $D(A)^{-1} = diag(1/a_{11}, 1/a_{22}, \ldots, 1/a_{nn})$ is the $n \times n$ diagonal matrix with the diagonal entries $1/a_{11}, 1/a_{22}, \ldots, 1/a_{nn}$. Since $\|I - D(A)^{-1}A\|_\infty \leq \max_i \frac{\sum_{j=1, j \neq i}^{n} |a_{ij}|}{|a_{ii}|} < \epsilon$, it follows:

Proposition 5.1 *If* A *is* ϵ*-diagonally dominant then* $\|I - D(A)^{-1}A\|_\infty < \epsilon$.

We generally will let $B^{(0)} = D(A)^{-1}$ be the initial approximation to the inverse of A in our Newton iterations. A is *diagonally dominant* if A is ϵ-diagonally dominant for $\epsilon < 1$. A is *strongly diagonally dominant* if A is ϵ-diagonally dominant for $\epsilon = 1 - 1/n^c$ for some $c > 0$. Define a quantity to be *extremely small* if it is of the form $\epsilon = 1/(n\|A\|)^{cn}$ for some large constant $c > 2$. (This definition of *extremely small* is required for our proofs of rapid convergence; but in practice, it is possible we could alter this definition of extremely small so that ϵ is a larger quantity, thus often decreasing the bit complexity of our algorithm.) A is *extremely diagonally dominant* if A is ϵ-diagonally dominant for an extremely small ϵ. Let a nonsingular A be *extremely well conditioned* if $\text{cond}A \leq 1 + \epsilon$ for an extremely small ϵ. Let $A \approx \tilde{A}$ if $\|A - \tilde{A}\|$ is extremely small. The following follows from basic norm proprieties (see also [K 81,K 87]):

Proposition 5.2 *Suppose* A *is extremely diagonally dominant. Then:*

- A *is extremely well conditioned,*
- *If* $A \approx \tilde{A}$ *then* $\tilde{A}$ *is also extremely diagonally dominant,*
- $A \approx D(A)$, *so* A *is, within extremely small relative error, a diagonal matrix (but of course* A *is not identical to* $D(A)$*).*

- $A^{-1} \approx D(A)^{-1}$, *so the inverse of* A *is, within extremely small relative error, also a diagonal matrix (but of course* A^{-1} *is not identical to* $D(A)^{-1}$*).*
- *If* $\|I - BA\|_\infty$ *is extremely small, and* $\|A\|_\infty \geq 1$, *then* $\|B\|_\infty \leq 2$.

5.2 Approximate Inverse of Diagonally Dominant Matrices

Let A be an $n \times n$ matrix. We will let $B^{(0)} = D(A)^{-1}$ be the initial approximation to the inverse of A in our Newton iterations. The Newton iteration (see Ben-Israel [B66], Ben-Israel and Cohen [BC 66], and Hotelling [H 43,Ht 43] for sequential Newton iteration and Pan and Reif [PR 85,PR 89] for parallel applications) generates a sequence of matrices $B^{(1)}, B^{(2)}, \ldots$ where
$B^{(k)} = B^{(k-1)}(2I - AB^{(k-1)})$.

Proposition 5.3 *If* A *is* ϵ*-diagonally dominant then* $\|I - B^{(k)}A\|_\infty < \epsilon^{2^k}$.

This follows since.

$$I - B^{(k)}A = I - B^{(k-1)}(2I - AB^{(k-1)})A$$

$$= (I - B^{(k-1)}A)^2.$$

Thus

$$\|I - B^{(k)}A\|_\infty = \|(I - B^{(k-1)}A)^2\|_\infty < (\epsilon^{2^{k-1}})^2 = \epsilon^{2^k}.$$

Thus if A is strongly or extremely diagonally dominant, then $k = O(\log n)$ Newton iterations suffice to compute the approximate inverse $B^{(k)}$ with error $2^{-n^{\Omega(1)}}$.

5.3 Newton Iterations with Inaccurate Input

We assume A is extremely diagonally dominant, so A is ϵ-diagonally dominant for some extremely small $\epsilon = 1/(n\|A\|)^{cn}$ for some sufficiently large $c > 8$. Let $E_k(\epsilon) = \epsilon^{2^k} 5^{2^k+2k}$. Note that $E_0(\epsilon) = \epsilon$, $(5E_{k-1}(\epsilon))^2 \leq E_k(\epsilon)$, and also $E_{c^* \log n}(\epsilon) = 2^{-n^{\Omega(1)}}$, for a sufficiently large constant $c^* > 1$.

Next we consider the case of Newton iteration where the input matrix A is not initially given with full accuracy, and instead we are provided on-line a sequence of approximates $\tilde{A}^{(0)}, \tilde{A}^{(1)}, \ldots$ where $\|A - \tilde{A}^{(k)}\|_\infty \leq E_k(\epsilon)$. Let $\tilde{B}^{(0)} = D(\tilde{A}^{(0)})^{-1} = diag(1/\tilde{a}_{11}, 1/\tilde{a}_{22}, \ldots, 1/\tilde{a}_{nn})$ where $\tilde{A}^{(0)} = [\tilde{a}_{ij}]$.

If $\tilde{A}^{(0)}$ is ϵ-diagonally dominant then by Proposition 5.1, $\|I - \tilde{B}^{(0)}\tilde{A}^{(0)}\|_\infty < \epsilon$. We will generate a sequence of matrices $\tilde{B}^{(0)}, \tilde{B}^{(1)}, \ldots$ where $\tilde{B}^{(k)} = \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})$.

Since A is assumed to be extremely diagonally dominant, then by assumption on the close approximation of A by the $\tilde{A}^{(k)}$, by Proposition 5.2 we have:

Proposition 5.4 *Each $\tilde{A}^{(k)}$ is also extremely diagonally dominant, and extremely well conditioned.*

Finally, we give an inductive proof of quadratic convergence of the entire iteration sequence:

Proposition 5.5 *If $\|A\|_\infty \geq 1$, A and $\tilde{A}^{(0)}$ are extremely ϵ-diagonally dominant, then we have $\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty < E_k(\epsilon)$.*

Proof: We use a proof by induction on k. Note that the Basis holds by assumption, since $\epsilon = E_0(\epsilon)$. Now for $k > 1$, assume the induction hypothesis: $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon)$.

By definition of the Newton Iterations,

$$I - \tilde{B}^{(k)}\tilde{A}^{(k)} = I - \tilde{B}^{(k-1)}(2I - \tilde{A}^{(k)}\tilde{B}^{(k-1)})\tilde{A}^{(k)}$$

$$= (I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})^2.$$

By assumption on the approximations to the $\tilde{A}^{(k)}$,

$$\|\tilde{A}^{(k)} - \tilde{A}^{(k-1)}\|_\infty \leq \|A - \tilde{A}^{(k)}\|_\infty + \|A - \tilde{A}^{(k-1)}\|_\infty \leq E_k(\epsilon) + E_{k-1}(\epsilon).$$

Since by Proposition 5.4, $\tilde{A}^{(k-1)}$ is extremely diagonally dominant and by the induction hypothesis, $\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty < E_{k-1}(\epsilon)$, and by assumption $\|A\|_\infty \geq 1$, Proposition 5.2 implies $\|\tilde{B}^{(k-1)}\|_\infty \leq 2$.

Also,

$$I - \tilde{B}^{(k-1)}\tilde{A}^{(k)} = (I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}) + \tilde{B}^{(k-1)}(\tilde{A}^{(k-1)} - \tilde{A}^{(k)}),$$

so we have the bound:

$$\|I - \tilde{B}^{(k-1)}\tilde{A}^{(k)}\|_\infty \leq \|I - \tilde{B}^{(k-1)}\tilde{A}^{(k-1)}\|_\infty + \|\tilde{B}^{(k-1)}\|_\infty \cdot \|\tilde{A}^{(k-1)} - \tilde{A}^{(k)}\|_\infty$$

$$\leq E_{k-1}(\epsilon) + 2(E_k(\epsilon) + E_{k-1}(\epsilon)) \leq 5E_{k-1}(\epsilon)$$

since $E_k(\epsilon) \leq E_{k-1}(\epsilon)$.

Thus we have:

$$\|I - \tilde{B}^{(k)}\tilde{A}^{(k)}\|_\infty = \|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})^2\|_\infty$$

$$\leq \|(I - \tilde{B}^{(k-1)}\tilde{A}^{(k)})\|_\infty^2 \leq (5E_{k-1}(\epsilon))^2 \leq E_k(\epsilon),$$

by definition of $E_k(\epsilon)$. ∎

Lemma 5.1 *If A is extremely diagonally dominant, then even with the above approximations to A, $k = O(\log n)$ Newton iterations still suffice to compute the approximate inverse $\tilde{B}^{(k)}$ with error $2^{-n^{\Omega(1)}}$.*

(Note: we can also show that Lemma 5.1 holds even if A is more moderately diagonally dominant, but the Lemma will suffice for the RF algorithm.)

5.4 Non-Pipelined Newton Iterations for RF

In this subsection we assume we are given a matrix A which is extremely diagonally dominant. We shall compute an approximation to the RF of A within error 2^{-n^c} for any given constant $c > 1$.

Recall again that the RF of A is a full binary tree of depth $\log n$. Each internal node is an $n/2^d \times n/2^d$ matrix $A_{d,\alpha}$. The RF of $A_{d,\alpha}$ is defined in terms of the RF of two $n/2^{d+1} \times n/2^{d+1}$ matrices, namely $A_{d+1,\alpha 1}$ and $A_{d+1,\alpha 0}$, and furthermore $A_{d+1,\alpha 1}$ is defined in terms of submatrices of $A_{d,\alpha}$ and the inverse of $A_{d+1,\alpha 0}$.

Fix some $k = c_1 \log n$, for some appropriate constant $c_1 > 2$. Note that the simplest and most obvious way we could use to compute the RF of A within error 2^{-n^c} is by proceeding up the recursion tree in $\log n$ stages. We first sketch this simple algorithm (we will give full details of a more efficient algorithm below). At each stage $d = 0, 1, \ldots, \log n$, we could compute for all nodes of depth d an approximation $\tilde{A}_{d,\alpha}$ of $A_{d,\alpha}$ within error $2^{-n^{c_1}}$. Then we could compute $B_{d,\alpha}^{(k)}$, which is a k-th Newton iteration matrix approximating the inverse of matrix $A_{d,\alpha}$ up to error ϵ^{2^k}, where $\epsilon = 1/(n\|A\|)^{c'n}$, for some sufficiently large constant $c' > 2$. We could use $B_{d,\alpha}^{(k)}$ to approximate $A_{d,\alpha}^{-1}$ within error $2^{-n^{c_1}}$. Since there are only $\log n$ levels in the RF, and since A is extremely diagonally dominant, the norm bounds given in Proposition 2.1 would ensure that the error of the overall approximate computation is at most 2^{-n^c}. At each stage we would need to compute approximations to two recursive inverses. Each such stage requires $O(\log^2 n)$ time and $O(P(n)) \geq 2^d O(P(n/2^d))$ processors for the required k repeated $n/2^d \times n/2^d$ matrix products for each of the 2^d nodes of depth d. The $\log n$ stages would require total parallel time $O(k \log^2 n) = O(log^3 n)$ using $\sum_{i=0}^{\log n} 2^d O(P(n/2^d)) \leq O(P(n))$ processors.

5.5 Pipelined Newton Iterations for RF

Next, we will improve on this $O(log^3 n)$ algorithm for the approximate RF, decreasing the time to $O(log^2 n)$ without an increase in processor bounds. As in Lemma 5.1, we consider the case of Newton iteration where the recursively defined matrices $A_{d,\alpha}$ are not initially given with full accuracy, and instead we are provided a sequence of improving approximations $\tilde{A}_{d,\alpha}^{(0)}, \tilde{A}_{d,\alpha}^{(1)}, \ldots$ where $\|A - \tilde{A}^{(k)}\|_\infty \leq E_k(\epsilon)$. In the special case $d = 0$, the approximation is exact, $A_{0,<>} = A$, where $<>$ is the empty string.

Given these approximants to A, we will generate a sequence of matrices $\tilde{B}_{d,\alpha}^{(0)}, \tilde{B}_{d,\alpha}^{(1)}, \ldots$ where

$\tilde{B}_{d,\alpha}^{(k)} = \tilde{B}_{d,\alpha}^{(k-1)}(2I - \tilde{A}_{d,\alpha}^{(k)} \tilde{B}_{d,\alpha}^{(k-1)})$. Thus, $\tilde{B}_{d,\delta}^{(k)}$ will approximate the inverse of $\tilde{A}_{d,\delta}^{(k)}$.

We will then use Lemma 5.1 to show quadratic convergence.

Newton Iteration Algorithm for an RF
INPUT: An extremely ϵ-diagonally dominant matrix A of size $n \times n$.
For each d, α for $0 \leq d \leq \log n$ and $\alpha \in \{0,1\}^d$, do in parallel:

1. INITIALIZATION: Let $\tilde{A}^{(0)}_{0,<>} = A$, where $<>$ is the empty string.
2. Let $k = c_1 \log n$, for some appropriate constant $c_1 > 2$.
3. For each $i = 1, \ldots, k$ do

   Loop Invariant: and

   - We have just computed $\tilde{A}^{(i-1)}_{d,\alpha}$ which approximates $A_{d,\alpha}$ with error $\|\tilde{A}^{(i-1)}_{d,\alpha} - A_{d,\alpha}\|_\infty < E_{i-1}(\epsilon')$ for $\epsilon' = 1/(n\|A\|)^{c'n}$ with $c' > 2$, and
   - we have just computed $\tilde{B}^{(i-1)}_{d,\alpha}$ which approximates $A^{-1}_{d,\alpha}$ with error specified by $\|I - \tilde{B}^{(i-1)}_{d,\alpha} A_{d,\alpha}\|_\infty < E_{i-1}(\epsilon')$.

   (a) If $i = 0$ then let $\tilde{B}^{(0)}_{d,\alpha} = diag(1/\tilde{a}_{11}, 1/\tilde{a}_{22}, \ldots, 1/\tilde{a}_{nn})$, where $\tilde{A}^{(0)}_{d,\alpha} = [\tilde{a}_{ij}]$.
   Else if $i > 0$ then let $\tilde{B}^{(i)}_{d,\alpha} = \tilde{B}^{(i-1)}_{d,\alpha}\,(2I - \tilde{A}^{(i)}_{d,\alpha}\tilde{B}^{(i-1)}_{d,\alpha})$.

   (b) For $d < \log n$, let

$$\tilde{A}^{(i)}_{d,\alpha} = \begin{bmatrix} \tilde{A}^{(i)}_{d+1,\alpha 0} & \tilde{X}^{(i)}_{d,\alpha} \\ \tilde{Y}^{(i)}_{d,\alpha} & \tilde{Z}^{(i)}_{d,\alpha} \end{bmatrix}.$$

   where $\tilde{A}^{(i)}_{d+1,\alpha 0}, \tilde{X}^{(i)}_{d,\alpha}, \tilde{Y}^{(i)}_{d,\alpha}, \tilde{Z}^{(i)}_{d,\alpha}$ are matrices each of size $n/2^{d+1} \times n/2^{d+1}$ and
   $\tilde{A}^{(i)}_{d+1,\alpha 1} = \tilde{Z}^{(i)}_{d,\alpha} - \tilde{Y}^{(i)}_{d,\alpha}\tilde{B}^{(i)}_{d+1,\alpha 0}\tilde{X}^{(i)}_{d,\alpha}$.

OUTPUT: Approximate RF $\{\tilde{A}^{(k)}_{d,\alpha}\}$, which approximates the RF of A within 2-norm error $\sqrt{n}2^{-n^{c_1}} < 2^{-n^c}$, for some $c > 1$

We now show that in the initial approximation $\{\tilde{A}^{(0)}_{d,\alpha}\}$ to the RF of A, the errors do not accumulate as much on the recursions. Recall that we have recursively defined: $A_{d+1,\alpha 1} = Z_{d,\alpha} - Y_{d,\alpha}A^{-1}_{d+1,\alpha 0}X_{d,\alpha}$. This implies:

Proposition 5.6 *(again follows from known properties of Schur complements and Proposition 2.1 Pan and Reif [PR 85, PR 92]) In the RF, for all $d, 0 \leq d \leq \log n - 1$ and $\alpha \in \{0,1\}^d$,*

- *if the $A_{d,\alpha}$ is symmetric positive definite, then so are $A_{d+1,\alpha 1}, A_{d+1,\alpha 0}$,*
- $\|A_{d+1,\alpha 1}\|, \|A_{d+1,\alpha 0}\| \leq \|A_{d,\alpha}\|$,
- $\|A^{-1}_{d+1,\alpha 1}\|, \|A^{-1}_{d+1,\alpha 0}\| \leq \|A^{-1}_{d,\alpha}\|$,

- $1/\|A_{d,\alpha}^{-1}\| \leq \min(\|A_{d+1,\alpha 1}\|, \|A_{d+1,\alpha 0}\|)$,
- $1/\|A_{d,\alpha}\| \leq \min(\|A_{d+1,\alpha 1}^{-1}\|, \|A_{d+1,\alpha 0}^{-1}\|)$,
- $cond(A_{d,\alpha 0}) \leq cond(A)$.

Since A is extremely diagonally dominant, by definition A is ϵ-diagonally dominant for $\epsilon = 1/(n\|A\|)^{cn}$ for some $c > 2$. By Proposition 5.2, A is extremely well conditioned. Since we have $cond(A_{d,\alpha 0}) \leq cond(A)$, each $A_{d,\alpha 0}$ is also extremely well conditioned.

Recall we have defined a quantity to be *extremely small* if it is of the form $\epsilon' = 1/(n\|A\|)^{c'n}$ for some $c' > 2$. By Proposition 5.2, A and all the matrices $A_{d,\alpha}$ in the RF of A are, at least within extremely small relative error, diagonal matrices. By Proposition 5.2, the inverse of such nearly diagonal matrices are also, with extremely small relative error, diagonal matrices. Since our initial approximations to the inverses of matrices occurring in the RF are themselves diagonal matrices, it follows by Proposition 5.2, that our initial approximations to its inverse gives an approximation to the RF, with extremely small relative error $\epsilon' = 1/(n\|A\|)^{c'n}$ for some $c' > 2$.

Lemma 5.2 *Let $\epsilon' = 1/(n\|A\|)^{c'n}$ for some $c' > 2$. In the Newton Iteration Algorithm for the RF, at each depth $d = 1, \ldots, \log n$, and for each $\alpha \in \{0,1\}^d$, on the first stage of Newton iteration, the initial approximation of the RF matrices at depth d, have extremely small ∞-norm error. In particular,*

$$\|A_{d,\alpha}^{(0)} - A_{d,\alpha}\|_\infty \leq \epsilon'$$

and

$$\|I - B_{d,\alpha}^{(0)} A_{d,\alpha}\|_\infty \leq \epsilon'.$$

Lemma 5.3 bounds the errors of the i-th approximate RF $\{A_{d,\alpha}^{(i)}\}$. It follows by induction directly from Lemma 5.1 using it as a basis for the induction Lemma 5.2.

Lemma 5.3 *For each $d = 1, \ldots, \log n$, and each $\alpha \in \{0,1\}^d$, the ∞-norm error for the i-th iteration is a $(\epsilon')^{2^i}$. In particular,*

$$\|A_{d,\alpha}^{(i)} - A_{d,\alpha}\|_\infty \leq E_i(\epsilon')$$

and

$$\|I - B_{d,\alpha}^{(i)} A_{d,\alpha}\|_\infty \leq E_i(\epsilon').$$

As in the Subsection 4.2, we use the technique of *stream contraction* to decrease the time to $O(\log^2 n)$ time without a processor penalty. To simplify notation, let us define $\Phi_{d,\alpha}^i$ to be the computation of $\tilde{A}_{d,\alpha}^{(i)}$ and $\tilde{B}_{d,\alpha}^{(i)}$, where this approximate computation is within error specified by Lemma 5.3.

The idea is to note that for each $d, 0 \leq d < \log n$, and each $\alpha \in \{0,1\}^d$ defining an internal node $\tilde{A}_{d,\alpha}$ of depth d, the computation $\Phi^i_{d,\alpha}$ depends only on the computations $\Phi^{i-1}_{d+1,\alpha 1}$ and $\Phi^{i-1}_{d+1,\alpha 0}$. Furthermore, the computation $\Phi^{i-1}_{d+1,\alpha 1}$ is defined in terms of submatrices of $\tilde{A}^{(i-1)}_{d,\alpha}$ and of $\tilde{A}^{(i-1)}_{d+1,\alpha 0}$. In particular, we need only to apply Newton iteration one more time to these approximated matrices.

This implies that we can pipeline the computation of $\Phi^i_{d,\alpha}$ just as we did in the previous section for $S^{(i)}_{d,\alpha}$ (except we use here $k = O(\log n)$) to reduce the parallel time from $\Omega(\log^3 n)$ to $O(\log^2 n)$ without a processor increase.

We pipeline the computation as follows: As our basis step $t = 0$ we have done the computation $\Phi^0_{d,\alpha}$ for all $d = 0, \ldots \log n$ and each $\alpha \in \{0,1\}^d$. We assume for our induction hypothesis that at time $t, 0 \leq t < k + \log n$ we have done the computation $\Phi^i_{d,\alpha}$ for all i, d, α where $0 \leq d \leq \log n$, $0 \leq i \leq t - d$, and $\alpha \in \{0,1\}^d$.

Then we apply the approximation RF formula and perform one more product at each node of the recursion tree to do computation $\Phi^{i+1}_{d,\alpha}$ by time $t+1$ for all i, d, α where $0 \leq d \leq \log n$ and $i = t + 1 - d$ and $\alpha \in \{0,1\}^d$.

Summarizing, we have (using a small constant factor slowdown to reduce the processor bounds from $O(P(n))$ to $P(n)$)

Lemma 5.4 *Given an $n \times n$ matrix A, which is extremely diagonally dominant, we can compute an approximation to the RF of A with error $2^{-n^{\Omega(1)}}$ in parallel time $O(\log^2 n)$ using $P(n)$ processors.*

6 Symmetric Matrices with Separable Graphs

A family of graphs is $s(n)$-*separable* if, given a graph G in the family of $n \geq O(1)$ nodes, we can delete a set of $s(n)$ nodes, separating G into subgraphs in the family of size $\leq 2/3n$ nodes. Clearly d-dimensional grids or dissection graphs are $s(n) = O(n^{\frac{d-1}{d}})$ separable, and Lipton and Tarjan [LT 79] showed planar graphs are $O(\sqrt{n})$-separable. A sparsity graph of a symmetric matrix has a vertex for every row (column) of the matrix and an edge wherever there is a non-zero entry of the adjacency matrix. Matrices with separable sparsity graphs arise naturally from VLSI circuit problems, structure problems, and discretization of 2- or 3-dimensional PDEs. For example, d-dimensional PDEs result in matrices whose sparsity graphs are d-dimensional grids or related dissection graphs which are $s(n) = O(n^{\frac{d-1}{d}})$ separable.

Let A be an $n \times n$ symmetric positive definite matrix A with a $s(n)$-separable sparsity graph. Lipton, Rose, and Tarjan [LRT79] and Pan and Reif [PR 85, PR 92] define a nested dissection ordering which is used to guide the Gaussian elimination process of the sparse matrix so as to minimize fill-in. We assume

the matrix has already been pre and post multiplied by a permutation matrix so that the resulting matrix A has the rows and columns in this order.

Using this ordering for sequential Gaussian elimination,

Lemma 6.1 *(Lipton, Rose, and Tarjan [LRT79]) Given an $n \times n$ symmetric positive definite matrix A with an $s(n)$-separable sparsity graph, a recursive LU factorization can be computed in sequential time $O(s(n)^3)$, exactly solving in the same time bound the problems DETERMINANT and LINEAR SYSTEM SOLVE.*

In the Parallel Nested Dissection algorithm of Pan and Reif [PR 85,PR 92], this nested dissection ordering is specified by a vertex partition sequence $V_0, V_1, \ldots, V_k$, for $k = O(\log n)$ which is used to guide the parallel elimination process. Let $n_0 = n$ and let $n_{d+1} = n_d - |V_d|$ for $d = 0, \ldots, k$. Assuming an $s(n)$-separable sparsity graph (see Lipton, Rose, and Tarjan [LRT79]), we define:

Nested Dissection(ND) RF Sequence Problem:

If possible, construct a sequence of matrices $A = A_0, A_1, A_2, \ldots, A_k$, where $k = O(\log n)$ and for $d = 0, 1, \ldots, k-1$, A_d is an $n_d \times n_d$ matrix which is partitioned as

$$A_d = \begin{bmatrix} W_d & X_d \\ Y_d & Z_d \end{bmatrix}$$

where

- $n_{d+1} = n_d - |V_d|$
- W_d, X_d, Y_d, Z_d are matrices,
- W_d is a block diagonal matrix of size $|V_d| \times |V_d|$ where each block is of size $s(n_d) \times s(n_d)$,
- X_d is of size $|V_d| \times n_{d+1}$,
- Y_d is of size $n_{d+1} \times |V_d|$ (if A is symmetric, then $Y_d = (X_d)^T$),
- Z_d is of size $n_{d+1} \times n_{d+1}$, and
- $A_{d+1} = Z_d - Y_d W_d^{-1} X_d$ is the *Schur complement.*

Note that Pan and Reif [PR 85,PR 92] show that the norm and condition bounds of Proposition 2.1 hold for any ND RF sequence.

Pan and Reif [PR 85,PR 92] show that an ND RF sequence can be computed in parallel time $O(\log^3 n)$ using $P(s(n))$ processors, also giving in the same time bounds solutions to the problems DETERMINANT and LINEAR SYSTEM SOLVE. The proofs in Pan and Reif [PR 85,PR 92] show that the work for ND RF sequence is dominated by the computation of $O(n/s(n_d))$ products and inverses of a sequence of dense submatrices of size $s(n_d) \times s(n_d)$ each requiring

$O(\log^2 n)$ time and $O(P(s(n_d))n/s(n_d))$ processors for $d = 0, \ldots, k = O(\log n)$. Thus the total time is $O(\log^3 n)$ and the processor bound is

$$\sum_{d=0}^{n} O(P(s(n_d))n/s(n_d)) \leq O(P(s(n))),$$

where $s(n)$ is of the form n^γ for $\gamma > 0$.

We now derive a parallel algorithm in this case, and reduce the parallel time from $\Omega(\log^3 n)$ to $O(\log^2 n)$ while still using $P(s(n))$ processors. We use a modified form of our (balanced) RF algorithm on the appropriately permuted input matrix.

The partitioning of blocks is again altered depending on the separator structure, following the usual techniques used in the above ND RF sequence. Again the nested dissection ordering is specified by the vertex partition sequence $V_0, V_2, \ldots, V_k$, for $k = O(\log n)$ which is used to guide the parallel elimination process. Let n_d be as defined above.

ND RF: If possible, compute a binary tree of depth $k = O(\log n)$ whose nodes are matrices. Each node of depth $d, 0 \leq d \leq k$ is a $n_{d,\alpha} \times n_{d,\alpha}$ matrix $A_{d,\alpha}$ where $\alpha \in \{0,1\}^d$ is a binary string of length d, and $n_{d,\alpha}$ is defined recursively below. The node is a leaf if $n_{d,\alpha} = 1$.

For each $d, 0 \leq d \leq k$, we specify the string 1^d to be *special*. For each $\alpha \in \{0,1\}^d$, if α is special and $n_{d,\alpha} > 1$, then we recursively decompose the matrix (using the ND RF sequence), setting $n_{d+1,\alpha 0} = |V_d|$ and $n_{d+1,\alpha 1} = n_{d+1} = n_d - |V_d|$. Otherwise, if $\alpha \in \{0,1\}^d$ is not special but $n_{d,\alpha} > 1$, then we recursively decompose the matrix evenly (using the RF defined at the start of this paper). Let $n_{d+1,\alpha 1} = \lfloor n_{d,\alpha}/2 \rfloor$, $n_{d+1,\alpha 0} = \lceil n_{d,\alpha}/2 \rceil$. If $d = 0$ then $A_{d,\alpha} = A$ is the root of the tree and α is the empty string. For $0 \leq d < k$, each matrix $A_{d,\alpha}$ of depth d with $n_{d,\alpha} > 1$ has exactly two children in the tree, $A_{d,\alpha 1}$ and $A_{d,\alpha 0}$ of depth $d+1$ which will be defined by recursion. In particular, for $d = 0, 1, \ldots, k-1$, $A_{d,\alpha}$ is an $n_{d,\alpha} \times n_{d,\alpha}$ matrix which is partitioned as

$$A_{d,\alpha} = \begin{bmatrix} A_{d+1,\alpha 0} & X_{d,\alpha} \\ Y_{d,\alpha} & Z_{d,\alpha} \end{bmatrix}$$

where

- $A_{d+1,\alpha 0}, X_{d,\alpha}, Y_{d,\alpha}, Z_{d,\alpha}$ are matrices,
- $A_{d+1,\alpha 0} = W_d$ is of size $n_{d+1,\alpha 0} \times n_{d+1,\alpha 0}$, (where if $\alpha = 1^d$ then $n_{d+1,\alpha 0} = |V_d|$ and $A_{d+1,\alpha 0}$ is a block diagonal matrix of size $|V_d| \times |V_d|$ with each block of size $s(n_d) \times s(n_d)$),
- $X_{d,\alpha}$ is of size $n_{d+1,\alpha 0} \times n_{d+1,\alpha 1}$,
- $Y_{d,\alpha}$ is of size $n_{d+1,\alpha 1} \times n_{d+1,\alpha 0}$ (if A is symmetric, then $Y_{d,\alpha} = (X_{d,\alpha})^T$),,

- $Z_{d,\alpha}$ is of size $n_{d+1,\alpha 1} \times n_{d+1,\alpha 1}$, and
- $A_{d+1,\alpha 1} = Z_{d,\alpha} - Y_{d,\alpha} A_{d+1,\alpha 0}^{-1} X_{d,\alpha}$ is the *Schur complement.*

Note: The above ND RF of A is defined to be very similar to the RF $A = A_0, A_1, A_2, \ldots, A_k$. In particular, $A_d = A_{d,1^d}$ for $d = 1, \ldots, k$. The only difference is that the ND RF also recursively factors the $W_d = A_{d+1,1^d 0}$ matrices appearing in the ND RF sequence. This takes $O(\log^2 n)$ time using $P(s(n))$ processors. Since this is an $s(n_d)$-block diagonal matrix of size $|V_d| \times |V_d|$ with each block of size $s(n_d) \times s(n_d)$), the processor bound is $O(P(s(n_d))n/s(n_d)) \leq P(s(n))$. These are recursively factored evenly (rather than use the separator structure), using the (balanced) RF defined at the start of this paper.

Note that the results of Pan and Reif [PR 85,PR 92] imply that the norm and condition bounds of proposition 5.6 hold also for any ND RF.

We again use the exact same pipelining technique used in Subsection 4.2 to decrease the parallel time bounds from $O(\log^3 n)$ to $O(\log^2 n)$. We use both the analysis of (balanced) RF defined at the start of this paper, as well as the analysis of ND RF sequence defined in Pan and Reif [PR 85,PR 92]. In particular, the proof of our $P(s(n))$ processor bounds follows exactly from the results of Pan and Reif [PR 85,PR 92] and from corollary 3.3 in this paper. Again in this sparse $s(n)$-separable case the work for ND RF is dominated by the computation of products and inverses of a sequence of dense submatrices of size $s(n_d) \times s(n_d)$ each requiring $O(\log^2 n)$ time and $O(P(n_d))$ processors for $d = 0, \ldots, k = O(\log n)$. However, due to our use of pipelining, all these inverses are computed simultaneously, so the total time is $O(\log^2 n)$ while the processor bound remains $\sum_{d=0}^{n} O(P(s(n_d)))$. This sum is $O(P(s(n)))$ if $s(n)$ is of the form n^γ for $0 < \gamma < 1$. Note that if the sparsity graph of A has constant degree or is planar, then the sparsity graph of $A^T A$ still has separator bound $O(s(n))$.

Theorem 6.1 *Let A be an $n \times n$ symmetric positive definite matrix with an $s(n)$-separable sparsity graph, where $s(n)$ is of the form n^γ for $0 < \gamma < 1$. If A is nonsingular, then an ND RF can be computed in parallel time $O(\log^2 n)$ time using $P(s(n))$ processors, and we can exactly solve the problems DETERMINANT and LINEAR SYSTEM SOLVE for A within the same parallel complexity.*

7 Banded Matrices

Recall that A is *b-banded* if $a_{ij} = 0$ for $2|j - i| + 1 > b$, so the non-zero entries occur only within a band of width b around the diagonal. Note that the sparsity graph of A has constant separator bound b, and also note that the sparsity graph of $A^T A$ also has constant separator bound $3b$. If $s(n)$ is a constant b, then the above sum of proof of Theorem 6.1 bounding the processors $\sum_{d=0}^{n} O(P(s(n_d)))$ is upper bounded by $O(P(b)n/b)$.

Corollary 7.1 *Let A be an $n \times n$ matrix which is b-banded. If A is symmetric positive definite, then an ND RF can be exactly computed in parallel time $O(\log n \log b)$ with $P(b)n/b$ processors. We can solve the problems DETERMINANT and also LINEAR SYSTEM SOLVE for nonsingular A (not necessarily symmetric positive definite) within the same parallel complexity, by computing the RF of $A^T A$, using the reductions of Lemma 2.2 and 2.4.*

8 Further Work

In a further paper [R 93b], we show that many structured linear systems can be solved exactly and efficiently in parallel, dropping processor bounds to linear, with polylog time bounds. We give much improved parallel algorithms for the exact solution and factorization, determinant, inverse, and finding rank of various structured matrices: in particular Toeplitz and matrices of bounded displacement rank. and their generalizations. These are the first polylog time bounds for these problems with linear processors. Our processor reduction is by far the major result of that subsequent paper; this processor reduction uses techniques specific to structured matrices (and not related to the enclosed paper) However, using pipelining techniques of the enclosed paper, we also decrease our time bounds, for solving these structured linear systems, to $O(\log^2 n)$ using $n(\log n)^\omega$ processors. In spite of this, we view this speedup due to pipelining as less consequential compared to our processor decrease for structured linear systems. We apply this result to efficient parallel algorithms for the following problems in the same parallel time and processor bounds: polynomial greatest common divisors (GCD) and extended GCD, polynomial resultant, Padé approximants of rational functions , and shift register synthesis and BCH decoding problems. This drops by a nearly linear factor the best previous processor bounds for polylog time parallel algorithms for these problems.

9 Acknowledgments

The author wishes to thank Shenfeng Chen, Chris Lambert, Arman Glodjo, K. Subramani, Aki Yoshida, Hongyan Wang, and Prokash Sinha for various edits and comments on the paper, Ken Robinson for excellent editorial assistance, and especially Deganit Armon for a very thorough technical reading of this paper and her insightful comments.

References

[AHU 74] A.V.Aho, J.E.Hopcroft, and J.D.Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA (1974).

[AR 92] D.Armon and J.Reif, *Space and time efficient implementations of parallel nested dissection*, Proc. SPAA'92 (1992), 344–352.

[B66] A.Ben-Israel, *A note on iterative method for generalized inversion of matrices*, Math. Comput. (1966), 439–440.

[BC 66] A.Ben-Israel and D.Cohen, *On iterative computation of generalized inverses and assorted projections*, SIAM J. Numer. Analysis (1966), 410–419.

[BM 75] A.Borodin and I.Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier (1975).

[GL 83] G.H.Golub and C.F.van Loan, *Matrix Computations*. Johns Hopkins Univ. Press (1989).

[H 43] H.Hotelling, *Some new methods in matrix calculation*. Ann. Math. Statist. (1943). 1–34.

[Ht 43] H.Hotelling, *Further points on matrix calculations and simultaneous equations*. Ann. Math. Statist. (1943), 440–441.

[J 92] J. JáJá. *An Introduction to Parallel Algorithms*. Addison-Wesley (1992).

[KP 91] E.Kaltofen and V.Pan. *Processor efficient parallel solution of linear systems over an abstract field*. Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures. (1991). 180-191.

[KP 92] E.Kaltofen and V.Pan. *Processor-efficient parallel solution of linear systems II The positive characteristic and singular cases*. Proc. 33rd Annual IEEE Symposium on F.O.C.S. (1992). 714–723.

[KS 91] E.Kaltofen and B.D.Saunders. *On Wiedemann's method of solving sparse linear systems*. in Proc. AAECC-5. Lecture Notes in Computer Science **536**. Springer-Verlag. (1991). 216–226.

[K 81] D.Knuth. *The Art of Computer Programming: Seminumerical Algorithms*. Addison-Wesley (1981).

[K 87] D.Kozen. *Fast parallel orthogonalization*. SIGACT News (1987). 47.

[LRT79] R.J.Lipton. D.J.Rose. and R.E.Tarjan. *Generalized Nested Dissection*. SIAM Jour. Num. Anal., **16** (1979) 346–358.

[LT 79] R.J.Lipton and R.E.Tarjan, *A Separator Theorem for Planar Graphs*. SIAM J. Applied Math. (1979). 177–189.

[MC 79] R.T.Moenck and J.H.Carter, *Approximate algorithms to derive exact solutions to systems of linear equations*, Proc. Eurosam, Lecture Notes in Computer Science (1979), 63–73.

[Pa 85] V.Pan, *Fast and efficient parallel algorithms for the exact inversion of integer matrices*, Proc. 5th Conf. FST and TCS Lecture Notes in Computer Science (1985), 504–521.

[Pan 87] V.Pan, *Complexity of Parallel Matrix Computations*, Theoretical Computer Science **54** (1987), 65–85.

[PR 85] V.Pan and J.Reif, *Efficient Parallel Solution of Linear Systems*, Proc. 17th ACM Symp. on Theory of Computing(STOC) (1985), 143–152.

[PR 89] V.Pan and J.H.Reif, *Fast and Efficient Parallel Solution of Dense Linear Systems*, Comp. Math. Applic. (1989), 1481–1491.

[PR 91] V. Pan and J.H.Reif, *The Parallel Computation of Minimum Cost Paths in Graphs by Stream Contraction*, Information Proc. Lett., **40**. (1991), pp. 79-83.

[PR 92] V.Pan and J.H.Reif, *Fast and Efficient Parallel Solution of Sparse Linear Systems*, SIAM Journal on Computing (1992).

Reif [R 93a]

[R 93a] J.H.Reif, ed., *Synthesis of Parallel Algorithms*, Morgan Kaufmann (1993).

[R 93b] *Work Efficient Parallel Solution of Toeplitz and other Structured Linear Systems*, Nov (1993).

[Sc 80] J.T.Schwartz, *Fast probabilistic algorithms for verification of polynomial identities*, J. ACM **27** (1980), 701–717.